الفصل العشرون

البرمجة النصية لـ HTTP

يحدد بروتوكول نقل النصوص التشعبية **Hypertext Transfer Protocol (HTTP)** كيف تطلب مستعرضات الإنترنت المستندات من ملقمات الويب وكيف ترسل محتويات النماذج إلى هذه الملقمات وكيف تستجيب ملقمات الويب لطلبات وعمليات الإرسال هذه. من الواضح أن مستعرضات الإنترنت تتعامل مع الكثير من **HTTP**. لكن **HTTP** هذا عادةً ما يكون خارج نطاق سيطرة البرامج النصية فهو يحدث عندما ينقر المستخدم على ارتباط أو يدخل نموذج. من الممكن، ولكن ليس دائماً، أن تقوم **JavaScript** ببرمجة **HTTP** نصياً.

يمكن بدء طلبات **HTTP** عندما يضبط برنامج نصي الخاصية **location** للغرض **Window** أو يستدعي الطريقة **submit()** للغرض **Form**. وفي كلا الحالتين، يحمل المستعرض صفحة جديدة في النافذة ويكتب فوق أي برنامج نصي يعمل فيها. يمكن لهذا النوع من برمجة **HTTP** أن يكون مفيداً في صفحات الويب متعددة الأطر لكنه ليس موضوع هذا الفصل. ندرس هنا كيف يمكن لكود **JavaScript** الاتصال مع ملقم الويب دون جعل مستعرض الإنترنت يعيد تحميل الصفحة المعروضة حالياً.

تملك العلامات **<img>** و **<iframe>** و **<script>** الخاصيات **src** وعندما يضبط برنامج نصي هذه الخاصيات على **URL**، يتم بدء طلب **HTTP GET** لتحميل محتويات **URL** هذا. لذا يمكن للبرنامج النصي أن يُمرر معلومات إلى ملقم الويب بترميز هذه المعلومات في جزء سلسلة الاستعلام من **URL** صورة وبضبط الخاصية **src** للعنصر **<img>**. يجب أن يعيد ملقم الويب صورة ما كنتيجة لهذا الطلب لكن يمكن لهذه الصورة أن تكون غير مرئية كصورة شفافة قياسها 1 بكسل في 1 بكسل على سبيل المثال*.

إن العلامات **<iframe>** إضافةٌ أحدث إلى **HTML** وهي أكثر استقراراً من العلامات **<img>** لأنه يمكن للملقم الويب أن يعيد نتيجة قابلة للفحص باستخدام البرنامج النصي بدلاً من إعادة ملف الصورة الثنائي. ولبرمجة **HTTP** نصياً باستخدام العلامة **<iframe>**، يرمز البرنامج النصي أولاً المعلومات للملقم

* تسمى الصور من هذا النوع أحياناً بعثرات الويب **Web Bugs** ولها سمعة سيئة بسبب الإساءة للخصوصية الذي يحدث عند استخدام هذه العثرات لإيصال المعلومات إلى ملقم غير ذاك الذي حُملت منه صفحة الويب. من الاستخدامات الشائعة والشرعية لعثرات الويب نذكر تعداد الزوار وتحليل الازدحام في مواقع الويب. عندما تبرمج صفحة ويب الخاصية **src** لصورة لإرسال معلومات ثانيةً إلى الملقم الذي حملت منه هذه الصفحة، فلا يوجد ما يدعونا للقلق إزاء خصوصية المستخدم.

الويب في URL ثم يضبط الخاصية src للعلامة <iframe> على URL هذا. ينشئ الملقم مستند HTML يحوي استجابته ويرسله ثانيةً إلى مستعرض الإنترنت الذي يعرضه في <iframe>. من غير الضروري أن يكون <iframe> مرئياً للمستخدم بل يمكن أن يكون مخفياً باستخدام CSS على سبيل المثال. يمكن للبرنامج النصي الوصول إلى استجابة الملقم عن طريق العبور في غرض المستند لـ <iframe>. لاحظ أن هذا العبور يخضع لقيود سياسة الأصل نفسه التي شرحناها في المقطع 13.8.2.

تملك العلامة <script> أيضاً الخاصية src التي يمكن ضبطها للتسبب بطلب HTTP ديناميكي. إن برمجة HTTP نصياً باستخدام العلامات <script> شيقٌ للغاية لأن التفسير يكون غير ضروري عندما تأخذ استجابة الملقم شكل كود JavaScript حيث أن مفسر JavaScript هو الذي ينفذ استجابة الملقم.

على الرغم من أن برمجة HTTP نصياً باستخدام العلامات <img> و <iframe> و <script> ممكنة إلا أن تنفيذها بشكل نقال مهمةٌ صعبة. يركز هذا الفصل على طريقة أخرى أكثر قوة لتحقيق ذلك. إن الغرض XMLHttpRequest مدعوم بشكل جيد في المستعرضات الحديثة ويؤمن وصولاً كاملاً إلى البروتوكول HTTP كما يمكنه بدء طلبات POST و HEAD إضافة إلى طلبات GET النظامية. يمكن للغرض XMLHttpRequest أن يعيد استجابة الملقم بشكل متزامن أو غير متزامن كما يمكنه أن يعيد المحتوى كنص أو كمستند DOM. يشكل الغرض XMLHttpRequest حجر الأساس لهيكلية تطبيقات الويب المعروفة بالاسم Ajax. سنشرح تطبيقات Ajax بعد توضيح كيفية عمل هذا الغرض.

سنعود إلى موضوع برمجة HTTP نصياً باستخدام العلامات <script> في نهاية هذا الفصل حيث سنوضح كيفية تحقيقها عندما لا يكون الغرض XMLHttpRequest متوفراً.

20.1 استخدام XMLHttpRequest

إن برمجة HTTP نصياً باستخدام XMLHttpRequest عملية مؤلفة من ثلاث مراحل:

- إنشاء غرض XMLHttpRequest.
- تحديد وإدخال طلب HTTP إلى ملقم الويب.
- استعادة استجابة الملقم بشكل متزامن أو غير متزامن.

تتضمن المقاطع الفرعية التالية معلومات إضافية عن كل مرحلة من هذه المراحل.

20.1.1 الحصول على غرض الطلب

لم يُضبط الغرض XMLHttpRequest بمعيار على الإطلاق وعملية إنشائه مختلفة في Internet Explorer عنها في بقية المنصات (لحسن الحظ أن واجهة برمجة التطبيقات التي تستخدم الغرض XMLHttpRequest هي نفسها على كل المنصات).

يتم إنشاء الغرض XMLHttpRequest في معظم المستعرضات باستدعاء بسيط للباني :

```
var request = new XMLHttpRequest();
```

لم يكن في IE قبل النسخة السابعة منه تابع باني ()XMLHttpRequest إذ أن XMLHttpRequest في IE 5 و IE 6 هو غرض ActiveX ويجب إنشاؤه بتمرير اسم الغرض إلى الباني ()ActiveXObject :

```
var request = new ActiveXObject("Msxml2.XMLHTTP");
```

لسوء الحظ فإن اسم الغرض مختلف في الإصدارات المختلفة من مكتبة XML HTTP الخاصة بـ Microsoft. لذا قد تضطر في بعض الأحيان، اعتماداً على المكتبات المثبتة على الزبون، إلى استخدام هذا الكود البديل :

```
var request = new ActiveXObject("Microsoft.XMLHTTP");
```

نرى في المثال 1-20 تابعاً خدمياً مستقلاً عن المنصة اسمه ()HTTP.newRequest يُستخدم لإنشاء أغراض XMLHttpRequest.

المثال 1-20 التابع الخدمي ()HTTP.newRequest.

```
// This is a list of XMLHttpRequest-creation factory functions to try
HTTP._factories = [
    function() { return new XMLHttpRequest(); },
    function() { return new ActiveXObject("Msxml2.XMLHTTP"); },
    function() { return new ActiveXObject("Microsoft.XMLHTTP"); }
];

// When we find a factory that works, store it here.
HTTP._factory = null;

// Create and return a new XMLHttpRequest object.
//
// The first time we're called, try the list of factory functions until
// we find one that returns a non-null value and does not throw an
// exception. Once we find a working factory, remember it for later use.
//
HTTP.newRequest = function() {
    if (HTTP._factory != null) return HTTP._factory();

    for(var i = 0; i < HTTP._factories.length; i++) {
        try {
            var factory = HTTP._factories[i];
            var request = factory();
            if (request != null) {
                HTTP._factory = factory;
                return request;
            }
        }
        catch(e) {
            continue;
        }
    }
```

```
    // If we get here, none of the factory candidates succeeded,
    // so throw an exception now and for all future calls.
    HTTP._factory
 = function() {
        throw new Error("XMLHttpRequest not supported");
    }
    HTTP._factory(); // Throw an error
}
```

20.1.2 إدخال الطلب

ما أن يتم إنشاء الغرض **XMLHttpRequest**، تكون الخطوة التالية هي إدخال الطلب إلى ملقم الويب وهذه الخطوة بحد ذاتها مؤلفة من عدد من الخطوات الفرعية. استدعِ أولاً الطريقة **open()** لتحديد **URL** الذي تطلبه والطريقة **HTTP** للطلب. تتم معظم طلبات **HTTP** باستخدام الطريقة **GET** التي تُنزِّل ببساطة محتويات **URL**. ومن الطرق المفيدة الأخرى **POST** التي تستخدمها معظم نماذج **HTML** وهي تسمح بتضمين قيم المتحولات المسماة كجزء من الطلب. إن **HEAD** هي طريقة **HTTP** مفيدة أخرى فهي تطلب من الملقم أن يُعيد الترويسات المرافقة لـ **URL**. إن هذا يمكّن البرنامج النصي على سبيل المثال، من التحقق من تاريخ تعديل المستند دون تنزيل محتوى المستند نفسه. حدد الطريقة و **URL** الطلب باستخدام الطريقة **open()**:

```
request.open("GET", url, false);
```

تضبط الطريقة **open()** في الحالة الافتراضية غرض **XMLHttpRequest** غير متزامن، وتمرير **false** في البارامتر الثالث يجعل هذه الطريقة تُحضر استجابة الملقم بشكل غير متزامن. إن الاستجابات غير المتزامنة مفضلة عادةً بيد أن تلك المتزامنة أسهل بقليل لذا سندرسها أولاً.

تقبل الطريقة **open()** إضافة إلى البارامتر الثالث الاختياري اسماً وكلمة مرور كبارامترين اختياريين رابع وخامس، ويُستخدم هذا البارامتران عند طلب **URL** من ملقم يطلب المصادقة.

إن الطريقة **open()** لا ترسل الطلب إلى ملقم الويب بل إنها تخزن ببساطة بارامترتها لتستخدم لاحقاً عندما يُرسل الطلب فعلياً. يجب أن تضبط أية ترويسات ضرورية للطلب قبل إرساله. وهذه بعض الأمثلة*:

```
request.setRequestHeader("User-Agent", "XMLHttpRequest");
request.setRequestHeader("Accept-Language", "en");
request.setRequestHeader("If-Modified-Since",
lastRequestTime.toString());
```

لاحظ أن مستعرض الإنترنت يضيف أوتوماتيكياً ملفات تعريف الارتباط ذات الصلة إلى الطلب الذي تبنيه وبالتالي يجب عليك تحديد الترويسة **"Cookie"** صراحةً فقط إذا أردت إرسال ملف تعريف ارتباط مزيف إلى الملقم.

* تخرج المعلومات التفصيلية عن البروتوكول **HTTP** عن نطاق هذا الكتاب. يمكنك الرجوع إلى مرجع عن **HTTP** للاطلاع على معلومات عن الترويسات التي يمكنك استخدامها عند تنفيذ طلب **HTTP**.

أخيراً وبعد إنشاء غرض الطلب واستدعاء الطريقة ()open وضبط الترويسات، أرسل الطلب إلى الملقم:

```
request.send(null);
```

إن بارامتر التابع ()send هو جسم الطلب ويكون null مع طلبات HTTP GET أما مع طلبات POST، فيجب أن يحوي بيانات النموذج التي سترسل إلى الملقم (راجع المثال 20-5). مرر حالياً null فقط (لاحظ هنا أن البارامتر null إجباري. إن الغرض XMLHttpRequest هو غرض على طرف الزبون ولا تتسامح طرقه، في Firefox على الأقل، مع البارامترات المحذوفة كما تفعل توابع JavaScript).

20.1.3 الحصول على استجابة متزامنة

إن غرض XMLHttpRequest لا يُخزن فقط تفاصيل طلب HTTP الذي نفذه بل إنه يمثل أيضاً استجابة الملقم. إذا مررت false في البارامتر الثالث للطريقة ()open فإن الطريقة ()send تكون متزامنة أي أنها تجمد ولا تعود حتى تصل استجابة الملقم*.

إن الطريقة ()send لا تعيد كود حالة، ويمكنك ما أن تتم العودة من هذه الطريقة أن تفحص كود حالة HTTP المُعاد من الملقم باستخدام الخاصية status لغرض الطلب. إن القيم الممكنة لهذا الكود معرفة في البروتوكول HTTP فالحالة 200 تعني أن الطلب كان ناجحاً وأن الاستجابة متوفرة، في حين أن الحالة 404 تدل على الخطأ "not found" الذي يحدث عندما يكون URL المطلوب غير متوفر.

يجعل الغرض XMLHttpRequest استجابة الملقم متوفرة كسلسلة محرفية عن طريق الخاصية responseText لغرض الطلب. إذا كانت الاستجابة مستند XML، فيمكنك الوصول إلى هذا المستند أيضاً على أنه غرض Document في DOM وذلك باستخدام الخاصية responseXML. لاحظ وجوب أن يحدد الملقم مستندات XML الخاصة به باستخدام نوع MIME المسمى "text/xml" كي يتمكن XMLHttpRequest من تفسير الاستجابة إلى غرض Document.

يبدو الكود الذي يلي الطريقة ()send عادةً كما يلي عندما يكون الطلب متزامناً:

```
if (request.status == 200) {
    // We got the server's response. Display the response text.
    alert(request.responseText);
}
else {
    // Something went wrong. Display error code and error message.
    alert("Error " + request.status + ": " + request.statusText);
}
```

* إن للغرض XMLHttpRequest مزايا قوية للغاية لكن واجهة برمجة تطبيقاته غير مصممة بشكل جيد. على سبيل المثال، من الأفضل أن تكون القيمة المنطقية التي تحدد السلوك المتزامن أو غير المتزامن بارامتراً للطريقة ()send.

يؤمن الغرض XMLHttpRequest، بالإضافة إلى أكواد الحالة ونص أو مستند الاستجابة، وصولاً إلى ترويسات HTTP المُعادة من ملقم الويب. تعيد الطريقة ()getAllResponseHeaders ترويسات الاستجابة ككتلة نص غير مفسرة في حين تعيد الطريقة ()getResponseHeader قيمة ترويسة مسماة، على سبيل المثال:

```
if (request.status == 200) {  // Make sure there were no errors
    // Make sure the response is an XML document
    if (request.getResponseHeader("Content-Type") == "text/xml") {
        var doc = request.responseXML;
        // Now do something with the response document
    }
}
```

ثمة مشكلة واحدة في استخدام XMLHttpRequest بشكل متزامن وهي أنه إذا توقف ملقم الويب عن الاستجابة فإن الطريقة ()send تجمد لوقت طويل، ويتوقف تنفيذ JavaScript ويبدو كما لو أن مستعرض الإنترنت قد علق (إن هذا يعتمد على المنصة بطبيعة الحال). إذا علق الملقم أثناء تحميل صفحة عادية، يمكن للمستخدم نقر الزر Stop في المستعرض وتجريب ارتباط آخر أو محدد موقع معلومات آخر. لكن لا يوجد زر Stop مع XMLHttpRequest إذ لا تقدم الطريقة ()send أي أسلوب لتحديد وقت انتظار أعظمي كما أن نموذج التنفيذ وحيد المسار لـ JavaScript على طرف الزبون لا يسمح لبرنامج نصي بمقاطعة XMLHttpRequest ما أن يتم إرسال الطلب.

يكمن الحل لهذه المشكلة في استخدام XMLHttpRequest بشكل غير متزامن.

20.1.4 التعامل مع استجابة غير متزامنة

لاستخدام الغرض XMLHttpRequest في النمط غير المتزامن، مرر true كبارامتر ثالث إلى الطريقة ()open (أو احذف ببساطة البارامتر الثالث فتُستخدم القيمة true في الحالة الافتراضية). إذا فعلت هذا فإن الطريقة ()send ترسل الطلب إلى الملقم ثم تعود حالاً. عندما تصل استجابة الملقم فإنها تصبح متوفرة في الغرض XMLHttpRequest عن طريق نفس الخواص المستخدمة في الحالة المتزامنة.

تشبه الاستجابة غير المتزامنة من الملقم نقرة الفأرة غير المتزامنة التي يقوم بها المستخدم إذ يجب تنبيه المستخدم بحدوثها. ويتم هذا باستخدام معالج حدث. يتم ضبط معالج الحدث مع الأغراض XMLHttpRequest على الخاصية onreadystatechange. يشير اسم هذه الخاصية إلى أن تابع معالج الحدث يتم استدعاؤه كلما تغيرت قيمة الخاصية readyState. إن readyState عبارة عن عدد صحيح يحدد حالة طلب HTTP وقيمه الممكنة واردة في الجدول 20-1. لا يعرف الغرض XMLHttpRequest ثوابتاً رمزية للقيم الخمسة المعروضة في هذا الجدول.

الجدول 1-20 قيم الخاصية readyState في الغرض XMLHttpRequest.

readyState	المعنى
0	لم تُستدعى ()open بعد.
1	أُستدعيت ()open لكن ()send لم تستدعَ.
2	أُستدعيت ()send لكن الملقم لم يستجب بعد.
3	يتم استقبال البيانات من الملقم. تختلف القيمة 3 readyState نوعاً ما بين Firefox و Internet Explorer. راجع المقطع 20.1.4.1.
4	اكتملت استجابة الملقم.

بما أن XMLHttpRequest يملك معالج حدث وحيد فقط، فإن هذا المعالج يستدعى مع كل الأحداث الممكنة. يُستدعى المعالج onreadystatechange مرةً واحدة عندما تُستدعى الطريقة ()open ومرة أخرى عندما تُستدعى الطريقة ()send. يُستدعى هذا المعالج مرة أخرى عندما تبدأ استجابة الملقم بالوصول ومرة أخرى أخيرة عندما تكتمل الاستجابة. وبخلاف معظم الأحداث في JavaScript على طرف الزبون، لا يتم تمرير أي غرض حدث إلى المعالج onreadystatechange. يجب أن تفحص الخاصية readyState للغرض XMLHttpRequest لتحديد السبب الذي أدى إلى استدعاء معالج الحدث. لسوء الحظ لا يمرر الغرض XMLHttpRequest كبارامتر إلى معالج الحدث أيضاً لذا يجب أن تتأكد من تعريف تابع معالج الحدث في مجال تغطية يمكّنه من الوصول إلى غرض الطلب. يبدو معالج حدث نموذجي لطلب غير متزامن كما يلي:

```
// Create an XMLHttpRequest using the utility defined earlier
var request = HTTP.newRequest();

// Register an event handler to receive asynchronous notifications.
// This code says what to do with the response, and it appears in a nested
// function here before we have even submitted the request.
request.onreadystatechange = function() {
    if (request.readyState == 4) {  // If the request is finished
        if (request.status == 200)  // If it was successful
            alert(request.responseText);     //  Display  the  server's
response
    }
}

// Make a GET request for a given URL. We don't pass a third argument,
// so this is an asynchronous request
request.open("GET", url);

// We could set additional request headers here if we needed to.

// Now send the request. Since it is a GET request, we pass null for
// the body. Since it is asynchronous, send() does not block but
// returns immediately.
request.send(null);
```

20.1.4.1 ملاحظات عن القيمة 3 للخاصية readyState

لم يُضبط الغرض XMLHttpRequest بمعيار على الإطلاق لذا فإن المستعرضات تختلف في معالجتها للقيمة 3 للخاصية readyState. على سبيل المثال يستدعي المستعرض Firefox أثناء عمليات التنزيل الكبيرة المعالج onreadystatechange عدة مرات في القيمة 3 لـ readyState لإعطاء معلومات عن تقدم عملية التنزيل. يمكن للبرنامج النصي أن يستخدم الاستدعاءات العديدة هذه لعرض مؤشر تقدم للمستخدم. من ناحية أخرى نجد أن Internet Explorer يفسر اسم معالج الحدث بشكل صارم إذ يستدعيه فقط عندما تتغير قيمة readyState فعلياً. إن هذا يعني أنه يُستدعى مرة واحدة فقط مع القيمة 3 لـ readyState بغض النظر عن حجم المستند الذي يتم تنزيله.

تختلف المستعرضات أيضاً في الجزء المتاح من استجابة الملقم في القيمة 3 لـ readyState. وعلى الرغم من أن القيمة 3 لـ readyState تعني أن جزءاً ما من الاستجابة قد وصل من الملقم، فإن توثيق Microsoft ينص صراحةً على أنه من الخطأ استعلام responseText في هذه الحالة. أما في المستعرضات الأخرى، فيمكن لاستعلام responseText أن يعيد الجزء المتاح من استجابة الملقم، لكن هذه الحقيقة غير موثقة صراحةً في هذه المستعرضات.

لسوء الحظ فإن أياً من مطوري المستعرضات الرئيسية لم يقدم بعد توثيقاً كافياً للأغراض XMLHttpRequest. لذا فإنه من الآمن بمكان، وحتى يتم وضع معيار أو تتم كتابة توثيق واضح، أن نتجاهل أية قيمة مختلفة عن 4 لـ readyState.

20.1.5 أمان XMLHttpRequest

يمكن لغرض XMLHttpRequest، وكجزء من سياسة الأصل نفسه (راجع المقطع 13.8.2)، أن يبدأ طلبات HTTP فقط إلى الملقم الذي جرى منه تنزيل المستند الذي يستخدم هذا الغرض. إن هذا القيد معقول ومنطقي ويمكنك تجاوزه إذا كنت مضطراً وذلك باستخدام برنامج نصي على طرف الملقم كملقم وكيل Proxy يجلب المحتويات التي تقع خارج الموقع.

إن لهذا القيد الأمني أثرٌ مهمٌ للغاية على XMLHttpRequest وهو أن XMLHttpRequest يصنع طلبات HTTP ولا يعمل مع مخططات URL أخرى. على سبيل المثال، لا يمكنه العمل مع URL التي تستخدم البروتوكول //:file. إن هذا يعني عدم إمكانية اختبار برامج XMLHttpRequest النصية من نظام الملفات المحلي بل لا بد من تحميل هذه البرامج التي تنوي اختبارها إلى ملقم ويب (أو أن تشغل ملقماً على حاسوبك الشخصي). كما يجب أن تُحمل هذه البرامج إلى مستعرض الإنترنت باستخدام HTTP وذلك كي تتمكن من بدء طلبات HTTP بمفردها.

20.2 أمثلة وبرامج خدمية عن XMLHttpRequest

عرضنا في بداية هذا الفصل التابع الخدمي ()HTTP.newRequest الذي يُستخدم لإنشاء غرض XMLHttpRequest لأي مستعرض. يمكننا تبسيط استخدام XMLHttpRequest أكثر عن طريق التوابع الخدمية. تتضمن المقاطع الفرعية التالية بعضاً من هذه التوابع.

20.2.1 توابع GET الخدمية الأساسية

يتضمن المثال 20-2 تابعاً بسيطاً للغاية يمكنه التعامل مع الاستخدامات الأكثر شيوعاً للغرض XMLHttpRequest. مرر إلى هذا التابع URL الذي تريد جلبه والتابع الذي يجب تمرير نص URL له.

المثال 20-2 التابع الخدمي ()HTTP.getText.

```
/**
 * Use XMLHttpRequest to fetch the contents of the specified URL using
 * an HTTP GET request. When the response arrives, pass it (as plain
 * text) to the specified callback function.
 *
 * This function does not block and has no return value.
 */
HTTP.getText = function(url, callback) {
    var request = HTTP.newRequest();
    request.onreadystatechange = function() {
        if (request.readyState == 4 && request.status == 200)
            callback(request.responseText);
    }
    request.open("GET", url);
    request.send(null);
};
```

أما المثال 20-3 فهو بديل بديهي يستخدم لجلب مستندات XML وتمرير تمثيلها المفسر إلى تابع إعادة الاستدعاء.

المثال 20-3 التابع الخدمي ()HTTP.getXML.

```
HTTP.getXML = function(url, callback) {
    var request = HTTP.newRequest();
    request.onreadystatechange = function() {
        if (request.readyState == 4 && request.status == 200)
            callback(request.responseXML);
    }
    request.open("GET", url);
    request.send(null);
};
```

20.2.2 جلب الترويسات فقط

من مزايا الغرض XMLHttpRequest أنه يمكنك من تحديد طريقة HTTP التي ستُستخدم. يطلب الطلب HTTP HEAD من الملقم أن يُعيد ترويسات URL معطى دون إعادة محتوى URL هذا. يمكن الإفادة من ذلك، على سبيل المثال، لفحص تاريخ تعديل مورد ما قبل تنزيله.

يبين المثال 20-4 كيفية تنفيذ طلب HEAD فهو يتضمن تابعاً لتفسير أزواج اسم/قيمة لترويسات HTTP وتخزينها كأسماءٍ وقيم لخواص غرض JavaScript. يتضمن هذا المثال أيضاً تابعاً لمعالجة الخطأ يُستدعى إذا أعاد الملقم 404 أو أي كود خطأ آخر.

المثال 20-4 التابع الخدمي ()HTTP.getHeaders.

```
/**
 * Use an HTTP HEAD request to obtain the headers for the specified URL.
 * When the headers arrive, parse them with HTTP.parseHeaders() and pass the
 * resulting object to the specified callback function. If the server returns
 * an error code, invoke the specified errorHandler function instead. If no
 * error handler is specified, pass null to the callback function.
 */
HTTP.getHeaders = function(url, callback, errorHandler) {
    var request = HTTP.newRequest();
    request.onreadystatechange = function() {
        if (request.readyState == 4) {
            if (request.status == 200) {
                callback(HTTP.parseHeaders(request));
            }
            else {
                if (errorHandler) errorHandler(request.status,
                                               request.statusText);
                else callback(null);
            }
        }
    }
    request.open("HEAD", url);
    request.send(null);
};

// Parse the response headers from an XMLHttpRequest object and return
// the header names and values as property names and values of a new object.
HTTP.parseHeaders = function(request) {
    var headerText = request.getAllResponseHeaders();  // Text from the server
    var headers = {}; // This will be our return value
    var ls = /^\s*/;  // Leading space regular expression
    var ts = /\s*$/;  // Trailing space regular expression

    // Break the headers into lines
    var lines = headerText.split("\n");
    // Loop through the lines
    for(var i = 0; i < lines.length; i++) {
        var line = lines[i];
        if (line.length == 0) continue;  // Skip empty lines
        // Split each line at first colon, and trim whitespace away
        var pos = line.indexOf(':');
```

```
        var name = line.substring(0, pos).replace(ls, "").replace(ts, "");
        var value = line.substring(pos+1).replace(ls, "").replace(ts, "");
        // Store the header name/value pair in a JavaScript object
        headers[name] = value;
    }
    return headers;
};
```

20.2.3 الطلب HTTP POST

يتم إدخال نماذج HTML في الحالة الافتراضية إلى ملقمات الويب باستخدام الطريقة HTTP POST. تُمرر البيانات مع طلبات POST إلى الملقم في جسم الطلب بدلاً من ترميزها في URL نفسه. وبما أن بارامترات الطلب ترمز في URL الطلب GET، فإن الطريقة GET مناسبةٌ فقط عندما لا يكون للطلب أي أثر جانبي على الملقم، أي عندما نتوقع أن تعيد طلبات GET المتكررة لنفس URL مع نفس البارامترات النتيجة نفسها. عند وجود ظواهر مرافقة للطلب (عندما يخزن الملقم بعض البارامترات في قاعدة بيانات)، فلا بد من استخدام الطلب POST بدلاً من GET.

يبين المثال 20-5 كيفية تنفيذ طلب POST مع غرض XMLHttpRequest. تستخدم الطريقة HTTP.post() التابع HTTP.encodeFormData() لتحويل خواص غرض إلى سلسلة محرفية يمكن استخدامها كجسم لطلب POST. تُمرر هذه السلسلة المحرفية فيما بعد إلى الطريقة XMLHttpRequest.send() وتصبح جسماً للطلب (يمكننا إلحاق السلسلة المحرفية المعادة من HTTP.encodeFormData() بمحدد موقع معلومات GET وذلك باستخدام محرف علامة الاستفهام للفصل بين URL والبيانات). يستخدم المثال 20-5 أيضاً الطريقة HTTP._getResponse() التي تفسر استجابة الملقم استناداً إلى نوعها وهي مضمنة في المقطع التالي.

المثال 20-5 التابع الخدمي HTTP.post().

```
/**
 * Send an HTTP POST request to the specified URL, using the names and values
 * of the properties of the values object as the body of the request.
 * Parse the server's response according to its content type and pass
 * the resulting value to the callback function. If an HTTP error occurs,
 * call the specified errorHandler function, or pass null to the callback
 * if no error handler is specified.
 **/
HTTP.post = function(url, values, callback, errorHandler) {
    var request = HTTP.newRequest();
    request.onreadystatechange = function() {
        if (request.readyState == 4) {
            if (request.status == 200) {
                callback(HTTP._getResponse(request));
            }
            else {
                if (errorHandler) errorHandler(request.status,
                                               request.statusText);
                else callback(null);
```

```
            }
        }
    }

    request.open("POST", url);
    // This header tells the server how to interpret the body of the request.
    request.setRequestHeader("Content-Type",
                             "application/x-www-form-urlencoded");
    // Encode the properties of the values object and send them as
    // the body of the request.
    request.send(HTTP.encodeFormData(values));
};

/**
 * Encode the property name/value pairs of an object as if they were from
 * an HTML form, using application/x-www-form-urlencoded format
 */
HTTP.encodeFormData = function(data) {
    var pairs = [];
    var regexp = /%20/g; // A regular expression to match an encoded space

    for(var name in data) {
        var value = data[name].toString();
        // Create a name/value pair, but encode name and value first
        // The global function encodeURIComponent does almost what we want,
        // but it encodes spaces as %20 instead of as "+". We have to
        // fix that with String.replace()
        var pair = encodeURIComponent(name).replace(regexp,"+") + '=' +
            encodeURIComponent(value).replace(regexp,"+");
        pairs.push(pair);
    }

    // Concatenate all the name/value pairs, separating them with &
    return pairs.join('&');
};
```

إن المثال **21-14** مثال آخر ينشئ طلب POST باستخدام غرض **XMLHttpRequest**. يستدعي ذلك المثال خدمة ويب وبدلاً من تمرير قيم النموذج في جسم الطلب، فإنه يُمرر نص مستند XML.

20.2.4 الاستجابات HTML و XML و JSON-Encoded

لقد تعاملنا مع استجابة الملقم على طلب HTTP في كل الأمثلة التي عرضناها حتى الآن على أنها قيمة نص بسيط. وهذا أمر جائزٌ تماماً ولا يوجد ما ينص على أن ملقمات الويب لا يمكنها إعادة المستندات التي نوع محتواها "text/plain". يمكن لكود JavaScript أن يفسر مثل هذه الاستجابات باستخدام الطرق String وأن يفعل بها ما يلزم.

يمكننا دائماً التعامل مع استجابة الملقم على أنها نص صرف حتى ولو كان نوع محتواها مختلفاً. إذا أعاد الملقم مستند HTML، على سبيل المثال، فيمكنك استعادة محتوى هذا المستند باستخدام الخاصية **responseText** ومن ثم استخدام هذا المحتوى لضبط الخاصية **innerHTML** لعنصر ما في المستند.

على أية حال، ثمة طرق أخرى للتعامل مع استجابة الملقم. ذكرنا في بداية هذا الفصل أنه إذا أرسل الملقم استجابة نوع محتواها "text/xml"، فيمكننا استعادة تمثيل مُفسر لمستند XML باستخدام الخاصية responseXML. إن قيمة هذه الخاصية هي غرض Document في DOM ويمكنك البحث والعبور فيه باستخدام طرق DOM.

لاحظ، على أية حال، أن استخدام XML كتنسيق بيانات قد لا يكون الخيار الأفضل دائماً. إذا أراد الملقم تمرير بيانات لتتم معالجتها بواسطة برنامج JavaScript نصي فإن ترميز هذه البيانات إلى XML على الملقم عملية غير فعالة. ومن الأفضل جعل الغرض XMLHttpRequest يفسر هذه البيانات إلى شجرة من عقد DOM ثم جعل البرنامج النصي يعبر في هذه الشجرة لاستخلاص البيانات. ثمة طريقة أسهل تقوم على جعل الملقم يرمز البيانات باستخدام أحرف أغراض ومصفوفات JavaScript وتمرير نص JavaScript المصدر إلى مستعرض الإنترنت. يفسر البرنامج النصي بعدها الاستجابة بتمريرها ببساطة إلى الطريقة eval() في JavaScript.

إن ترميز البيانات بصيغة أحرف أغراض ومصفوفات JavaScript يعرف باسم ترميز غرض JavaScript أو JavaScript Object Notation (JSON)*. نرى هنا ترميزي XML و JSON لنفس البيانات:

```
 <!-- XML encoding -->
<author>
  <name>Wendell Berry</name>
  <books>
    <book>The Unsettling of America</book>
    <book>What are People For?</book>
  </books>
</author>

// JSON Encoding
{
  "name": "Wendell Berry",
  "books": [
    "The Unsettling of America",
    "What are People For?"
  ]
}
```

يستدعي التابع HTTP.post() المعروض في المثال 20-5 التابع HTTP._getResponse() الذي يفحص الترويسة Content-Type لتحديد صيغة الاستجابة. إن المثال 20-6 هو تضمين بسيط للتابع HTTP._getResponse() يعيد مستندات XML كأغراض Document ويُقيم مستندات JavaScript أو JSON باستخدام الطريقة eval() ويعيد أي محتوى آخر كنص بسيط.

* يمكنك تعلم المزيد عن JSON بزيارة الارتباط http://json.org. تم تقديم فكرة JSON من قبل Douglas Crockford ويتضمن هذا الموقع ارتباطات إلى مرمزات ومفككات ترميز JSON لعدد كبير من لغات البرمجة. إن JSON ترميزٌ مفيدٌ للبيانات حتى ولو لم تكن تستخدم JavaScript.

المثال 20-6 التابع ()HTTP._getresponse.

```
HTTP._getResponse = function(request) {
    // Check the content type returned by the server
    switch(request.getResponseHeader("Content-Type")) {
    case "text/xml":
        // If it is an XML document, use the parsed Document object.
        return request.responseXML;

    case "text/json":
    case "text/javascript":
    case "application/javascript":
    case "application/x-javascript":
        // If the response is JavaScript code, or a JSON-encoded value,
        // call eval() on the text to "parse" it to a JavaScript value.
        // Note: only do this if the JavaScript code is from a trusted server!
        return eval(request.responseText);

    default:
        // Otherwise, treat the response as plain text and return as a string.
        return request.responseText;
    }
};
```

لا تستخدم الطريقة ()eval لتفسير بيانات JSON المُرمزة كما فعلنا في المثال 20-6 إلا إذا كنت واثقاً تماماً من أن ملقم الويب لن يرسل كود JavaScript تنفيذي خبيث بدلاً من بيانات JSON مُرمزة بشكل جيد. من البدائل الآمنة، استخدام مفكك ترميز JSON يفسر أحرف أغراض JavaScript يدوياً دون استدعاء ()eval.

20.2.5 انقضاء مهلة طلب

من نقاط الضعف في الغرض XMLHttpRequest كونه لا يؤمن أية طريقة لتحديد وقت انقضاء مهلة للطلب. إن نقطة الضعف هذه فاضحةٌ مع الطلبات المتزامنة. إذا علق الملقم فإن المستعرض يبقى جامداً في الطريقة ()send ويتوقف كل شيء عن العمل. لا تتأثر الطلبات غير المتزامنة بالجمود لأن الطريقة ()send فيها لا تعلق وبالتالي يمكن لمستعرض الإنترنت مواصلة معالجة أحداث المستخدم. افترض أن التطبيق ينفذ طلب HTTP مع غرض XMLHttpRequest عندما ينقر المستخدم على زر. من المفيد، لمنع الطلبات المتعددة، أن نلغي تفعيل هذا الزر حتى تصل الاستجابة. لكن ماذا إذا انهار الملقم أو فشل لسبب ما في الاستجابة للطلب؟ إن المستعرض لن يجمد لكن التطبيق سيجمد مع زر معطل.

لمنع هذا النوع من المشاكل، قد يكون من المفيد أن تضبط وقت انقضاء المهلة الخاص بك مستخدماً التابع ()Window.setTimeout عند تنفيذ طلبات HTTP. ستحصل في الحالة الطبيعية على الاستجابة قبل إطلاق معالج انقضاء المهلة، وعندها تستخدم التابع ()Window.clearTimeout لإلغاء وقت انقضاء المهلة هذا. من ناحية أخرى، إذا أُطلق معالج انقضاء المهلة قبل وصول الغرض XMLHttpRequest إلى القيمة 4 للخاصية readyState، فيمكنك إلغاء الطلب باستخدام الطريقة ()XMLHttpRequest.abort.XML.

من الأفضل بعدها أن تعلم المستخدم بفشل الطلب (ربما باستخدام الطريقة ()Window.alert). إذا عطلت الزر قبل تنفيذ الطلب فيجب أن تعيد تفعيله بعد انقضاء وقت المهلة.

يعرف المثال 7-20 التابع ()HTTP.get الذي يوضح تقنية وقت انقضاء المهلة هذه وهو يمثل نسخة مطورة من الطريقة ()HTTP.getText المعرفة في المثال 2-20 كما أنه يتضمن العديد من المزايا الواردة في الأمثلة السابقة بما فيها معالج خطأ وبارامترات طلب والطريقة ()HTTP._getResponse. يمكّن هذا المثال المستخدم من تحديد تابع إعادة استدعاء تقدم اختياري يُستدعى كلما استدعي المعالج onreadystatechange مع قيم غير القيمة 4 لـ readyState. يمكّن تابع إعادة استدعاء التقدم البرنامج النصي من عرض معلومات عن التنزيل للمستخدم وذلك في المستعرضات كـ Firefox التي تستدعي هذا المعالج عدة مرات في الحالة 3.

المثال 7-20 التابع الخدمي ()HTTP.get.

```
/**
 * Send an HTTP GET request for the specified URL. If a successful
 * response is received, it is converted to an object based on the
 * Content-Type header and passed to the specified callback function.
 * Additional arguments may be specified as properties of the options object.
 *
 * If an error response is received (e.g., a 404 Not Found error),
 * the status code and message are passed to the options.errorHandler
 * function. If no error handler is specified, the callback
 * function is called instead with a null argument.
 *
 * If the options.parameters object is specified, its properties are
 * taken as the names and values of request parameters. They are
 * converted to a URL-encoded string with HTTP.encodeFormData() and
 * are appended to the URL following a '?'.
 *
 * If an options.progressHandler function is specified, it is
 * called each time the readyState property is set to some value less
 * than 4. Each call to the progress-handler function is passed an
 * integer that specifies how many times it has been called.
 *
 * If an options.timeout value is specified, the XMLHttpRequest
 * is aborted if it has not completed before the specified number
 * of milliseconds have elapsed. If the timeout elapses and an
 * options.timeoutHandler is specified, that function is called with
 * the requested URL as its argument.
 **/
HTTP.get = function(url, callback, options) {
    var request = HTTP.newRequest();
    var n = 0;
    var timer;
    if (options.timeout)
        timer = setTimeout(function() {
                               request.abort();
                               if (options.timeoutHandler)
                                   options.timeoutHandler(url);
                           },
```

```
                            options.timeout);

    request.onreadystatechange = function() {
        if (request.readyState == 4) {
            if (timer) clearTimeout(timer);
            if (request.status == 200) {
                callback(HTTP._getResponse(request));
            }
            else {
                if (options.errorHandler)
                    options.errorHandler(request.status,
                                         request.statusText);
                else callback(null);
            }
        }
        else if (options.progressHandler) {
            options.progressHandler(++n);
        }
    }

    var target = url;
    if (options.parameters)
        target += "?" + HTTP.encodeFormData(options.parameters)
    request.open("GET", target);
    request.send(null);
};
```

20.3 Ajax والبرمجة النصية الديناميكية

يصف المصطلح Ajax هيكلية من تطبيقات الويب التي تؤدي دور HTTP المبرمج نصياً والغرض XMLHttpRequest (في الحقيقة، إن المصطلحين Ajax و XMLHttpRequest مترادفان في العديد من التطبيقات). إن Ajax هو اختصار (بأحرف ليست كلها كبيرة) لـ JavaScript و XML غير المتزامنة Asynchronous JavaScript and XML*. وضع هذا المصطلح Jesse James Garrett وظهر للمرة الأولى في مقالته التي نشرها في شباط من العام 2005 بعنوان "Ajax: A New Approach to Web Applications.". يمكنك العثور على هذه المقالة في الارتباط http://www.adaptivepath.com/publications/essays/archives/000385.php.

إن غرض XMLHttpRequest الذي تستند عليه Ajax متوفر في مستعرضات الإنترنت من Microsoft و Netscape/Mozilla قبل نشر مقالة Garrett بأربع سنوات بيد أنه لمن يحظى أبداً بمثل هذا الاهتمام. لقد تغير هذا في العام 2004 عندما أطلقت Google تطبيق البريد الالكتروني Gmail باستخدام XMLHttpRequest. لقد أدى هذا الإطلاق إضافة إلى مقالة Garrett إلى فتح الأبواب على مصراعيها أمام سيل جارف من الاهتمام بـ Ajax.

* إن الهيكلية Ajax قوية للغاية وإعطاؤها اسماً وحيداً أدى إلى ثورة في تصميم تطبيقات الويب. تبين فيما بعد، على أية حال، أن هذا المصطلح لا يوضح التقنيات التي تشكل تطبيقات Ajax. تستخدم كل برامج JavaScript على طرف الزبون معالجات الأحداث لذا فهي غير متزامنة. كما أن استخدام XML في تطبيقات Ajax مناسبٌ غالباً لكنه اختياريٌ دائماً. إن الخاصيات المميزة لتطبيقات Ajax هي استخدامها لـ HTTP المبرمجة نصياً لكن هذه الخاصيات لا تظهر في الاختصار.

إن الميزة الأساسية في تطبيق Ajax أنه يستخدم HTTP المبرمجة نصياً للاتصال مع ملقم ويب دون التسبب بإعادة تحميل الصفحات. وبما أن مقدار البيانات التي يتم تبادلها صغير غالباً وبما أن المستعرض ليس بحاجة إلى تفسير وتشكيل المستند (وأوراق النمط والبرامج النصية المرافقة له) فإن وقت الاستجابة يتحسن بشكل كبير ونحصل بالنتيجة على تطبيقات ويب تبدو كتطبيقات سطح المكتب التقليدية.

ومن المزايا الاختيارية في Ajax نذكر استخدام XML كترميز للبيانات المتبادلة بين الزبون والملقم. يشرح الفصل 21 كيفية استخدام JavaScript على طرف الزبون لمعالجة بيانات XML وتنفيذ استعلامات XPath وتحويلات XSLT لـ XML إلى HTML. تستخدم بعض تطبيقات Ajax التقنية XSLT لفصل المحتوى (بيانات XML) عن العرض (تنسيق HTML الملتقط كورقة نمط XSL). إن لهذه المقاربة مزايا إضافية تتمثل بتقليل حجم البيانات التي يجب نقلها من الملقم إلى الزبون وتقليل تفريغ التحويل من الملقم إلى الزبون.

من الممكن صياغة Ajax في ميكانيكية RPC*. يستخدم مطورو الويب في هذه الصياغة مكتبات Ajax منخفضة المستوى على طرفي الزبون والملقم لتسهيل الاتصال عالي المستوى بين الطرفين. لا يشرح هذا الفصل أياً من مكتبات RPC-over-Ajax لأنه يركز على التقنيات الأخفض مستوى التي تجعل Ajax يؤدي عمله على أكمل وجه.

إن Ajax بنية تطبيقات فتية وقد دعت مقالة Garrett في ختامها إلى ما يلي:

> "إن التحديات الكبرى في إنشاء تطبيقات Ajax ليست تقنية إذ أن تقنيات Ajax الجوهرية ناضجة ومستقرة ومفهومة بشكل جيد. بل إن التحديات التي تواجه مصممي هذه التطبيقات هي في نسيان ما نعرفه عن قيود الويب والبدء بتخيل مجال أوسع من الإمكانيات".

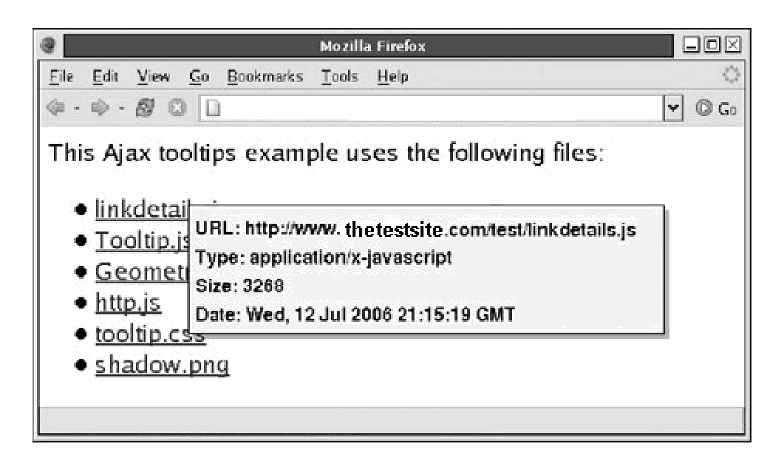
سيكون الأمر ممتعاً بالفعل.

20.3.1 مثال على Ajax

إن أمثلة XMLHttpRequest التي عرضناها حتى الآن في هذا الفصل كانت مجرد توابع خدمية توضح كيفية استخدام الغرض XMLHttpRequest وهي لم توضح السبب الذي قد يدفعك إلى استخدام هذا الغرض أو حتى ما يمكنك أن تفعل به. تفتح الهيكلية Ajax الباب أمام العديد من الإمكانيات التي سنبدأ باستكشافها معاً. إن المثال التالي بسيط لكنه يغطي بعضاً من ميزات الهيكلية Ajax.

* إن RPC اختصار لاستدعاء الطريقة البعيدة Remote Procedure Call وهي إستراتيجية تُستخدم في الحوسبة الموزعة لتسهيل الاتصال بين الزبون والملقم.

إن المثال 20-8 عبارة عن برنامج نصي مغمور يسجل معالجات الأحداث على ارتباطات في المستند بحيث تعرض هذه الارتباطات تلميحات شاشة عندما يحرك المستخدم الفأرة عليها. يستخدم هذا البرنامج الغرض XMLHttpRequest لتنفيذ طلب HTTP HEAD مع الارتباطات التي تشير إلى الملقم نفسه الذي جرى منه تحميل المستند. واستناداً إلى الترويسات المعادة، يستخلص البرنامج النصي نوع المحتوى وحجم وتاريخ تعديل المستند المرتبط ويعرض هذه المعلومات في تلميح الشاشة (لاحظ الشكل 20-1). وهكذا نجد أن تلميح الشاشة يعرض فكرة عامة عن وجهة الارتباط ويمكنه أن يساعد المستخدم على تقرير فيما إذا كان سينقر على هذا الارتباط أم لا.



الشكل 20-1 تلميح شاشة باستخدام Ajax.

يستند هذا الكود على الصف Tooltip الذي طورناه في المثال 16-4 (بيد أنه لا يحتاج إلى امتداد هذا الصف الذي طورناه في المثال 17-3). يستخدم هذا المثال أيضاً الوحدة النمطية Geometry من المثال 14-2 والتابع الخدمي HTTP.getHeaders() من المثال 20-4. يستخدم هذا المثال عدة طبقات من اللاتزامن وذلك في صيغة معالج الحدث onload ومعالج الحدث onmouseover ومؤقت وتابع إعادة الاستدعاء للغرض XMLHttpRequest لذا فإنه يحوي توابع معششة عميقاً.

المثال 20-8 تلميحات شاشة باستخدام Ajax.

```
/**
 * linkdetails.js
 *
 * This unobtrusive JavaScript module adds event handlers to links in a
 * document so that they display tool tips when the mouse hovers over them for
 * half a second. If the link points to a document on the same server as
 * the source document, the tool tip includes type, size, and date
 * information obtained with an XMLHttpRequest HEAD request.
```

```
 *
 * This module requires the Tooltip.js, HTTP.js, and Geometry.js modules
 */
(function() {  // Anonymous function to hold all our symbols
    // Create the tool tip object we'll use
    var tooltip = new Tooltip();

    // Arrange to have the init() function called on document load
    if (window.addEventListener) window.addEventListener("load", init, false);
    else if (window.attachEvent) window.attachEvent("onload", init);

    // To be called when the document loads
    function init() {
        var links = document.getElementsByTagName('a');
       // Loop through all the links, adding event handlers to them
        for(var i = 0; i < links.length; i++)
            if (links[i].href) addTooltipToLink(links[i]);
    }

    // This is the function that adds event handlers
    function addTooltipToLink(link) {
        // Add event handlers
        if (link.addEventListener) {  // Standard technique
            link.addEventListener("mouseover", mouseover, false);
            link.addEventListener("mouseout", mouseout, false);
        }
        else if (link.attachEvent) {  // IE-specific technique
            link.attachEvent("onmouseover", mouseover);
            link.attachEvent("onmouseout", mouseout);
        }

        var timer; // Used with setTimeout/clearTimeout

        function mouseover(event) {
            var e = event || window.event;
           // Get mouse position, convert to document coordinates, add offset
            var x = e.clientX + Geometry.getHorizontalScroll() + 25;
            var y = e.clientY + Geometry.getVerticalScroll() + 15;

            // If a tool tip is pending, cancel it
            if (timer) window.clearTimeout(timer);

            // Schedule a tool tip to appear in half a second
            timer = window.setTimeout(showTooltip, 500);

            function showTooltip() {
                // If it is an HTTP link, and if it is from the same host
                // as this script is, we can use XMLHttpRequest
                // to get more information about it.
               if (link.protocol == "http:" && link.host == location.host) {
                    // Make an XMLHttpRequest for the headers of the link
                    HTTP.getHeaders(link.href, function(headers) {
                        // Use the headers to build a string of text
                        var tip = "URL: " + link.href + "<br>" +
                            "Type: " + headers["Content-Type"] + "<br>" +
                            "Size: " + headers["Content-Length"] + "<br>" +
                            "Date: " + headers["Last-Modified"];
                        // And display it as a tool tip
                        tooltip.show(tip, x, y);
                    });
```

```
            }
            else {
                // Otherwise, if it is an off-site link, the
                // tool tip is just the URL of the link
                tooltip.show("URL: " + link.href, x, y);
            }
        }
    }

    function mouseout(e) {
        // When the mouse leaves a link, clear any
        // pending tool tips or hide it if it is shown
        if (timer) window.clearTimeout(timer);
        timer = null;
        tooltip.hide();
    }
}
})();
```

20.3.2 تطبيقات الصفحة الواحدة (Single-Page Applications)

إن تطبيق الصفحة الواحدة، وكما يدل اسمه، هو تطبيق ويب مقاد باستخدام JavaScript يتطلب تحميل صفحة واحدة فقط. إن بعض تطبيقات الصفحة الواحدة لا تحتاج مطلقاً للتخاطب مع الملقم ما أن يتم تحميلها. ومن الأمثلة عليها نذكر ألعاب DHTML التي تؤدي فيها التفاعلات مع المستخدم إلى تعديلات مبرمجة نصياً على المستند المحمل.

يفتح الغرض XMLHttpRequest والهيكلية Ajax الباب واسعاً أمام العديد من الإمكانيات ويمكن لتطبيقات الويب استخدام هذه التقنيات لتبادل البيانات مع الملقم مع احتفاظها بهويتها كتطبيقات صفحة واحدة. يمكن لتطبيق الويب المصمم بهذه الطريقة أن يحوي مقداراً صغيراً من كود بدء الإقلاع المكتوب باستخدام JavaScript إضافة إلى نافذة Splash بسيطة تُعرض أثناء تهيئة التطبيق. ما أن يتم عرض النافذة Splash يمكن لكود بدء الإقلاع أن يستخدم الغرض XMLHttpRequest لتنزيل كود JavaScript الفعلي للتطبيق الذي سينفذ فيما بعد باستخدام الطريقة ()eval. بعدها يستلم كود JavaScript زمام المبادرة مُحملاً البيانات حسب الحاجة باستخدام XMLHttpRequest ومستخدماً DOM لتشكيل هذه البيانات كـ DHTML تعرض للمستخدم.

20.3.3 البرمجة النصية البعيدة (Remote Scripting)

سبق مصطلح البرمجة النصية البعيدة في تاريخ ظهوره المصطلح Ajax بحوالي أربع سنوات وهو ببساطة اسم أقل جاذبية للفكرة نفسها أي استخدام HTTP المبرمجة نصياً لإنشاء تداخل أكثر قوة (ووقت استجابة أفضل) بين الزبون والملقم. لقد شرحت إحدى المقالات ذائعة الصيت من Apple في العام 2002 كيفية استخدام العلامة <iframe> لإنشاء طلبات HTTP مبرمجة نصياً إلى ملقم الويب (راجع الارتباط http://developer.apple.com/internet/webcontent/iframe.html). تشير هذه المقالة إلى أنه إذا

أرسل ملقم الويب ملف HTML مع علامات <script> فيه، فإن كود JavaScript المحتوى في هذه العلامات يُنفذ باستخدام المستعرض ويمكنه استدعاء طرق معرفة في النافذة التي تحوي العلامة <iframe>. يمكن للملقم بهذه الطريقة إرسال أوامر مباشرة إلى زبونه بصيغة تعليمات JavaScript.

20.3.4 تحذيرات من Ajax

تعاني الهيكلية Ajax، حالها حال أية هيكلية أخرى، من بعض مواطن الخلل. يشرح هذا المقطع مواطن الخلل هذه كي تكون حذراً منها لدى تصميمك لتطبيقات Ajax.

إن أول مواطن الخلل هو التغذية الخلفية المرئية. فعندما ينقر المستخدم على ارتباط تشعبي تقليدي فإن مستعرض الإنترنت يقدم تغذية خلفية للدلالة على أن محتوى هذا الارتباط قيد الجلب، وهذه التغذية الخلفية تظهر حتى قبل أن يكون المحتوى جاهزاً للعرض كي يعرف المستخدم أن المستعرض يعمل على تنفيذ طلبه. على أية حال، عند تنفيذ طلب HTTP باستخدام XMLHttpRequest فإن المستعرض لا يقدم أية تغذية خلفية. يتسبب كمون الشبكة Network Latency حتى في الاتصالات عالية السرعة بتأخير ملحوظ بين تنفيذ طلب HTTP واستقبال الاستجابة على هذا الطلب. لهذا السبب، من المهم في تطبيقات Ajax تأمين نوع من التغذية الخلفية المرئية (كتحريك DHTML بسيط. راجع الفصل 16) أثناء انتظار استجابة على XMLHttpRequest.

لاحظ أن المثال 20-8 لا يكترث لهذه النصيحة في تأمين تغذية خلفية مرئية. والسبب هو أن المستخدم في هذا المثال لا يقوم بأي عمل يبدأ طلب HTTP بل إن الطلب يبدأ كلما حرك المستخدم الفأرة على ارتباط تشعبي. إن المستخدم لا يطلب من التطبيق صراحةً أن يقوم بأي عمل لذا فإنه لا يتوقع أي تغذية خلفية.

أما ثاني مواطن الخلل التي تعاني منها Ajax فهو متعلق بـ URLs إذ أن تطبيقات الويب التقليدية تنتقل من حالة إلى الحالة التالية بتحميل صفحات جديدة ولكل صفحة محدد موقع معلومات فريد. إن هذا لا يصح مع تطبيقات Ajax لأن URL في شريط الموضع لا يتغير عندما يستخدم تطبيق Ajax برمجة HTTP النصية لتنزيل وعرض محتوى جديد. قد يرغب المستخدمون بوضع إشارة مرجعية على حالة محددة ضمن التطبيق لكنهك سيفاجؤون بعدم إمكانية تحقيق ذلك باستخدام تقنية الإشارات المرجعية في المستعرض. لا يمكن للمستخدمين أيضاً قص ولصق محدد موقع المعلومات من شريط موضع المستعرض.

إن موطن الخلل هذا وحلوله موضحة في تطبيق خرائط Google (http://local.google.com). إذ ينقل كم كبير من البيانات بين الزبون والملقم لدى تكبير وتمرير الخريطة لكن URL المعروض في المستعرض لا يتغير على الإطلاق. لقد حلت Google هذه المشكلة بتضمين ارتباط "link to this page" على كل صفحة. إن النقر على هذا الارتباط يولد URL للخريطة المعروضة حالياً ويعيد تحميل الصفحة

باستخدام هذا الـ URL. وما أن تتم إعادة التحميل ، يمكننا وضع إشارة مرجعية على حالة الخريطة أو إرسال الارتباط التشعبي بالبريد الالكتروني إلى صديق. إن الدرس الذي يجب أن يتعلمه مطورو Ajax هو أنه من المفيد أن يكونوا قادرين على تغليف حالة التطبيق في URL وأن URLs هذه يجب أن تكون متاحة للمستخدمين عند الضرورة.

إن موطن الخلل الثالث في Ajax يتعلق بالزر Back. فالبرامج النصية التي تستخدم XMLHttpRequest تتجاوز ميكانيكية تاريخ المستعرض وذلك بالتحكم بـ HTTP بشكل مستقل عن هذا المستعرض. إن المستخدمين معتادون على الإبحار في الويب باستخدام الزرين Back و Forward. إذا استخدم تطبيق Ajax برمجة HTTP النصية لعرض أجزاء أساسية من المحتوى الجديد وحاول المستخدمون استخدام هذين الزرين للإبحار ضمن التطبيق ، فسيجدون أن الزر Back يخرج بالمستعرض من التطبيق بدلاً من عرض جزء المحتوى الذي عُرض مؤخراً!

جرى القيام بالعديد من المحاولات لحل مشكلة الزر Back وذلك بخداع المستعرض ودفعه إلى حشر URLs في تاريخه. لكن هذه التقنيات تستخدم كوداً مخصصاً للمستعرض وهي غير مرضية أبداً. وحتى عندما تعمل بشكل جيد، فإنها تخرب نموذج Ajax وذلك بدفع المستخدم إلى الإبحار بإعادة تحميل الصفحات بدلاً من الاعتماد على HTTP المبرمجة نصياً.

برأيي أن مشكلة الزر Back غير خطرة على الإطلاق ويمكن تقليلها بالتصميم المدروس للويب. إن عناصر التطبيق التي تبدو كارتباطات تشعبية يجب أن تسلك سلوك ارتباطات تشعبية كما يجب أن تقوم بإعادة تحميل حقيقي للصفحة. الأمر الذي يجعلها خاضعةً لميكانيكية تاريخ المستعرض وهو ما يتوقعه المستخدم. وعلى العكس من ذلك فإن عناصر التطبيق التي تعتمد على HTTP المبرمجة نصياً وغير الخاضعة لميكانيكية تاريخ المستعرض، يجب ألا تبدو كارتباطات تشعبية. تأمل تطبيق خرائط Google مرة أخرى. عندما ينقر المستخدم بالفأرة ويحرك للتمرير في الخريطة فهو لا يتوقع أن يقوم الزر Back بإلغاء هذا التمرير.

يجب تجنب استخدام كلمات مثل "forward" و "back" في عناصر التحكم بالإبحار ضمن تطبيقات Ajax. إذا استخدم التطبيق واجهة شبيهة بالمعالج تحوي زرين Next و Previous ، على سبيل المثال، فيجب أن يستخدم هذا التطبيق تحميل الصفحة التقليدي (بدلاً من XMLHttpRequest) لعرض الشاشة التالية أو الشاشة السابقة لأن المستخدم يتوقع في هذه الحالة أن يعمل الزر Back في المستعرض بنفس طريقة عمل الزر Previous في التطبيق.

يجب عدم الخلط بين الزر Back في المستعرض والميزة Undo في التطبيق إذ يمكن لتطبيقات Ajax أن تضمن الخيارات undo/redo الخاصة بها إذا كان هذا مفيداً للمستخدم، لكن يجب تذكر أن هذا يختلف عما يقدمه الزران Back و Forward.

20.4 برمجة HTTP نصياً باستخدام العلامات <script>

إن الغرض XMLHttpRequest هو غرض ActiveX في Internet Explorer 5 و 6. يعطل المستخدمون أحياناً برمجة ActiveX النصية في Internet Explorer لأسباب أمنية وفي هذه الحالة تكون البرامج النصية عاجزة عن إنشاء غرض XMLHttpRequest. يمكن عند الضرورة تنفيذ طلبات HTTP GET باستخدام العلامتين <script> و <iframe>. على الرغم من أن إعادة تضمين كل وظيفة XMLHttpRequest بهذه الطريقة غير ممكن*، فمن الممكن إنشاء نسخة من التابع الخدمي HTTP.getText() تعمل بدون برمجة ActiveX النصية.

يمكننا توليد طلبات HTTP بسهولة وذلك بضبط الخاصية src للعلامة <script> أو <iframe>. إن ما يخدع أكثر هو استخلاص البيانات التي تريدها من تلك العناصر دون أن يعدل المستعرض هذه البيانات. تتوقع العلامة <iframe> تحميل مستند HTML فيها وإذا حاولت تنزيل محتوى ملف نصي بسيط في <iframe> فستجد أن نصك يُحول إلى HTML. زد على ذلك أن بعض نسخ Internet Explorer لا تُضمن وبشكل مناسب المعالج onload أو المعالج onreadystatechange للعلامات <iframe> الأمر الذي يُصعّب العمل أكثر.

تستخدم المقاربة المعروضة هنا العلامة <script> وبرنامج نصي على طرف الملقم. حيث نخبر البرنامج النصي على طرف الملقم بـ URL الذي نريد محتواه وبالتابع على طرف الزبون الذي يجب أن يُمرر له المحتوى. يجلب البرنامج النصي على طرف الملقم محتويات URL المرغوب ويرمزها كسلسلة محرفية في JavaScript (قد تكون طويلة)، ويعيد برنامج نصي على طرف الزبون يُمرر هذه السلسلة المحرفية إلى التابع المحدد. وبما أن البرنامج النصي على طرف الزبون يُحمل في علامة <script> فإن التابع المحدد يُستدعى أوتوماتيكياً على محتويات URL عند اكتمال عملية التنزيل.

نرى في المثال 20-9 تضميناً لبرنامج نصي مناسب على طرف الملقم بلغة البرمجة النصية PHP.

المثال 20-9 jsquoter.php.

```
<?php
// Tell the browser that we're sending a script
header("Content-Type: text/javascript");
// Get arguments from the URL
$func = $_GET["func"];        // The function to invoke in our js code
$filename = $_GET["url"];     // The file or URL to pass to the func
$lines = file($filename);     // Get the lines of the file
$text = implode("", $lines); // Concatenate into a string
// Escape quotes and newlines
$escaped = str_replace(array("'", "\"", "\n", "\r"),
                       array("\\'", "\\\"", "\\n", "\\r"),
                       $text);
```

* قد يتطلب البديل الكامل لـ XMLHttpRequest استخدام بريمج Java.

```
// Output everything as a single JavaScript function call
echo "$func('$escaped');"
?>
```

يستخدم التابع على طرف الزبون في المثال 20-10 البرنامج النصي على طرف الملقم **jsquoter.php** في المثال 20-9 ويعمل بشكل مشابه للتابع **HTTP.getText()** الوارد في المثال 20-2.

المثال 20-10 التابع الخدمي HTTP.getTextWithScript().

```
HTTP.getTextWithScript = function(url, callback) {
    // Create a new script element and add it to the document.
    var script = document.createElement("script");
    document.body.appendChild(script);

    // Get a unique function name.
    var funcname = "func" + HTTP.getTextWithScript.counter++;

    // Define a function with that name, using this function as a
    // convenient namespace. The script generated on the server
    // invokes this function.
    HTTP.getTextWithScript[funcname] = function(text) {
        // Pass the text to the callback function
        callback(text);

        // Clean up the script tag and the generated function.
        document.body.removeChild(script);
        delete HTTP.getTextWithScript[funcname];
    }

    // Encode the URL we want to fetch and the name of the function
    // as arguments to the jsquoter.php server-side script. Set the src
    // property of the script tag to fetch the URL.
    script.src = "jsquoter.php" +
                 "?url=" + encodeURIComponent(url) + "&func=" +
                 encodeURIComponent("HTTP.getTextWithScript." + funcname);
}

// We use this to generate unique function callback names in case there
// is more than one request pending at a time.
HTTP.getTextWithScript.counter = 0;
```